INTELLIGENT ADDRESSING AND ROUTING OF DATA VIA THE RTI FILTER CONSTRUCTS

Larry F. Mellon
Science Applications International Corporation

KEYWORDS
Interest Management, Multicast, Filtering, HLA

ABSTRACT

The HLA RTI is in part intended to increase the scalability of an exercise by replacing the DIS broadcast paradigm with one of optimal data delivery to the minimal subset of simulation hosts as is possible. The basic mechanism provided, class-based subscription, supports only a limited increase in scale. A secondary mechanism, filtering, is defined to provide additional scaling support. An API to a filtering technique referred to as *filter space* is proposed for inclusion into the baseline RTI API definition. This paper first defines the key scaling issues that affect the filtering problem, the classes of filtering techniques proposed to date within the AMG, and how filtering may be used in optimizing the data flow between simulation hosts. An analysis of the filter space API's support for other filtering techniques is given, and how differing routing algorithms may be employed.

INTRODUCTION

A new standard for DoD simulation programs has been proposed by DMSO and the Architecture Management Group (AMG). The High-Level Architecture (HLA) is intended to foster interoperability and standardized infrastructure where possible across all simulation programs, with especial focus given to distributed simulations.

The HLA defines a single logical component to support execution, object, and time management, as well as synchronized data exchanges between simulations (*federates*). The draft specification of this Run-Time Infrastructure (RTI) component provides a robust definition of all management services, but has not yet fully addressed how the overall system overhead scales in proportion to the number of entities in the exercise. Further scalability issues within the RTI must be addressed in terms of the number of platforms executing a given federation and the data exchange rates between federates.

This paper first defines the basic scaling issues within distributed simulation which are related to known data filtering techniques. The basic classes of filtering techniques under discussion in the AMG are introduced, and implementation options are summarized. The proposed scaling mechanism of the RTI (filter space) is then analyzed in terms of both scalability in and of itself, as well as the suitability of the filter space API for supporting other classes of filtering mechanisms.

SCALABILITY SUPPORT VIA THE RTI

The primary mechanism for scalability in the RTI is *object subscription*, which allows federates to specify what classes of data are required at their individual machine hosts. This knowledge allows the RTI to improve the inter-machine transmission protocols from non-scaling broadcast protocols (ala DIS) to various multicast transmission mechanisms with better scaling characteristics.

A secondary mechanism, referred to as filtering, is defined in concept but not implementation.[i] RTI filters allow federates to more tightly specify what data is required at their machine hosts based on the current value of individual attributes. While this concept certainly allows for a scaleable system, the actual implementation of the RTI filters (and the intrinsic scalability of the exercise) will have a strong impact on the performance of a federation. Other mechanisms to support scalability, such as DIS dead reckoning algorithms and variable fidelity data are not directly supported by the RTI, but must rather be addressed by the federation. The actual implementation of such federate-level techniques and their interaction with RTI filtering concepts and implementations will also greatly affect overall system scalability.

Filter Definition

It is assumed that large federations with high performance requirements must perform work beyond that of simple class-based object subscription to reduce the amount of incoming data at each federate. This concept is broadly referred to as filtering, and is now specified in the RTI draft specification under Data Distribution Management. As defined in [Mellon96], filtering may exist both within federates and within the RTI, where RTI-level filters are used to reduce the packets arriving at any given federate host, and federate-specific filters are used to further reduce the data set before detailed processing.

Filter space has been proposed as a mechanism within the RTI to implement filtering for a broad range of federations. Further, it is proposed as an interface and implementation capable of supporting other filtering techniques by federations whose data sharing characteristics may be better exploited by a custom filtering scheme.

SCALABILITY ISSUES AFFECTING FILTERING

As outlined above, a federation's scalability may be broadly defined as the ratio by which system overhead and model computation increase as the number of entities in the system increases. If the system overhead increases at an unacceptable rate[ii] as the number of entities increases, there is an upper bound on the number of entities that may be supported in an exercise. Similarly, if the amount of model computation increases at too high a rate as the number of entities simulated increases, an upper-bound exists.

Secondary scaling issues relate to the number of hosts in the federation and the volume of data exchanges between federations (i.e. is the implementation of the RTI scaleable). The following terms define the key scalability issues for a federation and their relationship to filtering. Note that this analysis does not factor in the effects on scalability of differing time management schemes, which may have a substantial impact on scalability and performance. Such effects are expected to be captured in the DMSO-sponsored white paper [Steinman96].

System Overhead

The primary system overhead for federations is expected to be the sharing of entity state data across large numbers of federate hosts. This is due to the large cost of transmitting and receiving network packets on current UNIX-based platforms, the number

[i] A proposed implementation known as in general as *filter space* is under investigation by DMSO. An API called *routing spaces* has been added to the RTI specification, but was not available in time for detailed review. No significant functional changes are expected to the filter space concept from the new API.

[ii] The level of acceptability will vary across federations. A geometric increase in overhead may be acceptable if the federation is small, a linear increase may be unacceptable if the federation is large

of entities and federate hosts in the system, and the frequency at which some entity state data changes occur. While the bandwidth of the underlying network must also be taken into consideration, previous experiments from the RITN project indicate that bandwidth will not be the dominating factor [Calvin95]. The majority of the proposed experiments will evaluate the effectiveness of various techniques in reducing the number of network accesses (both transmission and reception) done by machine hosts.

RTI-level filters are expected to be a primary technique in reducing system overhead due to network I/O caused by entity state changes.

Intrinsic Scalability

Standard simulation theory states that as each entity changes state, it must evaluate its new potential interactions with the full remaining set of entities in the simulation. Such an evaluation technique scales poorly, as the number of evaluations increases geometrically in proportion to the number of entities. Thus the *intrinsic scalability* of the system is considered to be low.

In practice, modelers will often optimize the problem of entity interaction evaluations with entity-specific knowledge, such as sensor type, predictable motions patterns of entities, relative positioning of entities, and similar techniques. By use of such knowledge, the modeler may either sort all entity state data for efficient evaluation (quad trees and similar approaches), or evaluate entity interactions with increasingly complex operations, where simple operations are used to remove the majority of entities from complex evaluations. Any technique which reduces the number of entity interaction evaluations required may be said to be increasing the intrinsic scalability of the system. Note that many of the common techniques employed are not suitable for use in distributed systems, due to the centralized nature of the algorithms.

In a distributed simulation, filtering via network multicast is an excellent way of exploiting intrinsic scalability. However, if the exercise exhibits poor intrinsic scalability, no filtering mechanism will have a significant impact on performance, as (by definition) the majority of data must be routed to the majority of hosts.

Given the large system overhead incurred by the sharing of large volumes of entity state data in a distributed system, the intrinsic scalability of any given exercise will severely impact that exercise's runtime performance, i.e. if the exercise design has poor intrinsic scalability, the system overhead will quickly exceed practical limits as the numbers of entities increases.

Data Transmission Optimizations

After the set of entity interaction evaluations has been reduced to the minimum set possible, some number of theoretical data exchanges between entities exists. A further technique may be used to increase the intrinsic scalability of an exercise by minimizing the actual number of network transmissions required to support the theoretical data exchanges. This class of optimizations is referred to as Data Transmission Optimizations (DTOs).

A DTO is defined as any technique that is used to optimize the network transmissions of shared state (or public attributes) in a distributed simulation. DTOs allow the federation designer to exploit the known characteristics of FOM data and how it is used, and thus are specified by each federation.

DTOs are not required for the exchange of model data, nor are they used to communicate information between models. They are used to only to automatically minimize the network transmissions required to keep the subset of federation shared state at each host sufficiently up-to-date.

Examples Of DTO Functionality

Restricting the amount of shared state to be distributed. Example: *interest declaration* (state the minimum set of data required by each simulation).

Organize data for efficient access. Example: *sectorization* (divide the battlespace into geographic sectors to help determine which objects are co-located).

Describe characteristics of the data itself. Example: *predictive contracts* (Dead Reckoning, scripts, etc, which predict how a data item changes over time).

Describe how data is used. Example: *variable resolution* (the amount of uncertainty an object is able to tolerate in a data item's value).

These improve the theoretical minimal data flow requirements for a federation. By exploiting the data requirements of the federates, the actual number of transmissions required to keep the system consistent is minimized, thus lowering the load on filters.

Locality of Reference

Another technique used to optimize the data flow of an exercise is *locality of reference.* If two entities are required to examine each other's state on a regular basis, and the entities are being modeled on machine hosts at opposite ends of the WAN, the system overhead incurred is large. If the entities are being modeled on machine hosts on the same LAN, the system overhead is lower. Correspondingly, if the entities are being modeled on the same machine host, system overhead is minimized. Mapping entities to hosts to exploit locality of reference will impose a lower load on the infrastructure, dependent on the infrastructure ‘s implementation approach.

Locality of reference is controlled by the initial allocation of modeled entities to machine hosts in the overall system, and later optimized by the *dynamic migration* of entities as they change state over time, and thus change potential localities of reference.

Note that locality of reference is a subset of the full load balancing problem: equal allocation of workload and balancing of CPU/memory resources are also important to scalability. However, filtering is

most affected by locality of reference, in that filters may be used to 'trap' local data from escaping localized areas of the system. See also: hierarchical architectures, below.

Hierarchical Architectures

A high-level architecture defines major system components in terms of their basic functionality, classes of interfaces, and the connectivity between components. A physical system architecture is defined as the mapping of high-level components to physical instances of hardware and/or software, and the connectivity between the physical components. Note that several different physical architectures may be used to implement the same conceptual architecture.

A hierarchical architecture in this context is used to refer to physical architectures which have similar elements grouped together to share resources and/or to lower system overhead. Such architectures tend to have good scaling behaviour, as no one component of the architecture must deal with large numbers of clients. For example, an architecture with 10,000 clients to a given server component creates a potential bottleneck in the system. A hierarchical architecture will break that central component into 10 individual components, each with 1000 clients: leaving each small server with less work to perform than the single large server with 10,000 clients. If required, the actions of the 10 servers are then coordinated to mimic the behaviour of a single server.

Examples of hierarchical physical architectures include shared caching mechanisms in the parallel-processing hardware community, where more than one processor shares a single cache [Chaiken91]. Such systems are designed to exploit locality of reference, where many processes share data. Efficiency gains are realized by two means:

- lower system overhead in bringing required data to the clients (a shared data item is brought to the local cache only once, not once per reading client).
- lower system overhead overall (data shared only by local clients does not leave the cache, and thus does not consume inter-cache resources nor interfere with other caches and/or clients of other caches by filling remote caches with non-used data).

It is important to note that such hierarchical architectures consume additional overhead to maintain and synchronize the levels within the hierarchy. However, the gain in scalability by removing bottlenecks is expected to exceed the additional cost incurred. Also note the dependence of many such systems on locality of reference. The overall system must be carefully engineered to increase locality.

With specific reference to distributed simulations, three natural hierarchical levels seem to exist:

- local federate host
- local area network
- wide area network.

A number of proposals exist in the community to exploit these levels with hierarchical services and/or shared caches. For example, a federate host may wish to cache shared data referred to by its local clients. A LAN may have a cache of data representing the set of required data for all local federate hosts. Across the entire federation, a shared conceptual service may be represented by a number of self-coordinated components, one per LAN. Another potential class of hierarchical optimization is 'clustering', where entities who tend to exchange data only with each other are executed on the same LAN, thus keeping non-global data

1) off the WAN
2) off of other LANs
3) off of other hosts

One potential use of filters is to restrict passage of data at each hierarchical point (i.e. exploit locality of reference).

Achieved Scalability

While intrinsic scalability gives the measure of how well the model may scale, *achieved scalability* gives the measure of how well the exercise infrastructure does scale for a given federation's level of intrinsic scalability. In other words, intrinsic scalability is the measure of how much data is required to be shared at each host, and achieved scalability is the measure of how efficient the infrastructure was in sharing said data.

Using again a standard DIS exercise as an example, techniques such as compression and packet-bundling are used to increase the efficiency of the infrastructure, and thus improve the system's achieved scalability.

Achieved scalability is used to determine if the amount of system overhead a given infrastructure implementation consumes is excessive or close to optimal. Two baselines are of use in measuring the achieved scalability: standard DIS, and a theoretical best case implementation, which has zero cost in exchanging data. The use of DIS is an obvious baseline, due to the broad use cases which exist, and its use of a broadcast protocol. The theoretical best case is used to establish an upper bound on an implementation's efficiency. A good measure of performance is how close to optimal efficiency did a given implementation achieve, and thus may be used to measure the effectiveness (and cost) of differing filter implementations.

Flow Control

Two forms of flow control may exist in interest-managed systems: *source quenching* and *interest quenching*. Source quenching in traditional network load management systems involves the detection of network overload, followed by a network signal to hosts that are producing the largest volumes of data (or have exceeded their provisioning requests). The hosts are then responsible for 'backing off' their data output rate, either unilaterally by low-level protocols, or via application-level decisions. Interest quenching is very similar: in network overload conditions, hosts that are consuming the largest volumes of data are

signaled to 'back off' by lowering the extent of their interest statements (i.e. reducing the scope of data requested by their filters).

Scalability Summary

Using a standard DIS exercise as a basis for comparison (where all entity state data is shared at all hosts), a maximum of several hundred entities may be realistically modeled. This maximum must be increased by one to two orders of magnitude for the RTI to be considered a success. Clearly, a careful analysis of the federations models and data exchange patterns must be performed to ensure a sufficiently high level of intrinsic scalability before any filtering technique may succeed. Standard examples of pathological cases include: wide-area viewers with metre-level resolution, and all entities within sensor range of all other entities.

As the issues listed above illustrate, efficient filtering techniques are a necessary but not sufficient condition for a scaleable federation. To summarize, a scaleable federation must:

- Reduce the occurrence of entity interaction evaluations to a minimum.

(*maximize intrinsic scalability)*

- Implement the models and the infrastructure in a scaleable fashion .

(*evenly distribute workload*)

- Optimize the theoretical data flows between entities, via DTOs, load balancing, ...

(*minimize communication load*)

- Implement transport mechanism (i.e. RTI and filters) in an efficient, scaleable fashion.

(*minimize communication cost*)

AMG FILTERING BACKGROUND

The AMG created a filter working group to further refine the notional filters specified in the early RTI drafts. The notional filters were required to execute locally to the calling federate's workstation (due to the inclusion of local state information and similar reasons). While this was considered adequate for federations with small amounts of shared state, a potential scaling problem was identified for federations with certain classes of data.

Ideally, data that did not pass a federate's filter set would not be sent to that federate, thus avoiding unnecessary consumption of federate resources servicing the network. In this context, the network itself is used as a first-order filter to prevent irrelevant data from arriving. However, the draft RTI specification was only able to perform this function in a limited manner. The basic difficulty was identified as a data routing problem, where the RTI lacked sufficient information to route attribute updates to the minimum set of required hosts. For certain federations, hosts were in danger of receiving far more data than they actually required. The following example was used.

Consider a federation with 10,000 platform-level entities in an exercise: 5,000 tanks and 5,000 jeeps. Each entity is represented by a single RTI-level object, consisting of an (x,y) coordinate. Two object classes are defined: 'tanks', and 'jeeps'.

With the then current RTI specification, any given federate may only subscribe to a class of objects, thus in the above example a federate may only subscribe to

- jeeps AND tanks,
- ONLY jeeps,
- ONLY tanks.

To continue the example, if a federate is only interested in "tanks within 10km of my_tank's current_position", a RTI filter may be defined to check the (x,y) values of incoming tank objects against the current (x,y) of "my_tank". Thus the RTI will only deliver to the federate tank objects which meet the filter. If only one tank fits the filter, then only one tank is passed up to the federate.

However, a key scaling issue is how the RTI can determine what tank objects meet the federate's filter. In the above example, the RTI must conceptually evaluate all 5,000 tank objects against the filter. As the filter contains references to dynamically changing data (the current positions of all remote tanks and the current position of the subscribing tank), the RTI must at the least evaluate each tank object against the filter when any tank objects change in value (i.e. an update_attribute is done). To do so, the RTI must bring all changed tank objects to the filter for evaluation. Note the initial condition of filters being resident on the owning federate's host. The RTI may filter out all jeep objects via the network, as the federate subscribed only to tank objects. However, all tank objects must be brought to the filter. Given that there are 5,000 tanks in the example case, and a DIS-like update rate of one per second, the RTI must bring 5,000 updates to the filter each second. At an approximate CPU cost of 150 to 500 microseconds per network access (as reported by the STOW SEID and JPSD projects), the upper bounds cost of maintaining our simple example filter is 500 usec * 5,000 = 2,500,000 usec / sec.

While the above numbers are approximate, they clearly show the scaling problem: a federate's host CPU is in danger of being swamped well above its capacity simply supporting the RTI filter.

FILTER PROBLEM DEFINITION

As the above example shows, it is not sufficient for the RTI filters to simply prevent unwanted data from being processed by a federate -- if the federate's host is receiving all data prior to the filter operation, little savings are realized. For our purposes then, filters must have two properties: first, they must prevent unwanted data from reaching the federate, and second, they must also prevent as much unwanted data as possible from reaching the host of the federate. The most straightforward mechanism to prevent unwanted data from arriving at a federate's host is simply not to send it to that host. To accomplish this task, two operations must be performed:

- *Addressing the Data*: i.e. obtaining the minimal list of federates requiring any given update_attribute.
- *Routing the Data*: i.e. moving the data from source to sink(s) with minimal cost to the system.

Given that each federate has specified a set of filters which describe what data does not meet its needs, the potential exists to invert the filters and determine what data needs to be routed to each federate (i.e. addressing the data). Thus only data that will pass each federate's filters will be routed to that federate.

The translation of federate-defined filters into identities of sources and sinks is conceptually a simple task, however the distributed nature of the problem complicates implementation choices considerably. The primary difficulty is that filters are defined by the sink and change over time, but the routing of data must occur at the source. Thus within the implementation the filtering mechanism must have some synchronization technique (either static or dynamic) between sources and (potential) destinations.

While the language used to specify filters must first permit the translation process from filter into routing-level information, a secondary consideration is the ease of use by the federation developer. Filters must be as simple as possible to define and modify, while still providing sufficient information to be used in the addressing process. It is postulated that many different techniques to specify filters are possible, each of which with differing interfaces, execution costs and benefits to the underlying routing mechanism.

ADDRESSING OF DATA

Addressing versus Routing

In a distributed system, filters may be categorized as obtaining Addressing information, i.e. who needs what subset of data items (one source per data item, N sinks). Multicast groups may be categorized as the Routing mechanism, i.e. how to get data from source to sink(s) with minimal cost. Overall costs to the system includes both cost of Addressing and Routing at each point in the system: source / network / sink. Note the independence between Addressing and Routing -- once addresses are obtained, more than one routing method may be employed.

Type-based Addressing

As the basic RTI subscription mechanism only understands static information (such as class and attribute types), that is all the information available for use in routing decisions. For example, one federate subscribes to 'tank' data, another federate publishes 'tank' data, and the RTI routes the 'tank' data from source to destination. But note that no value information is used -- routing is done strictly on the type of the data. While this will result in sufficient routing optimizations for many federations, it clearly does not scale well enough for very large federations.

The AMG filter working group agreed that federations with large numbers of objects and high performance requirements needed to be able to give the RTI more information to use in the routing decision. In the ideal case, only data which would pass the federate-defined value-based filters would be sent to that particular federate. This would greatly improve the potential scalability of the RTI -- the federate host in the above example would not receive 5,000 packets per second, but rather only the single tank object which passed its filter.

Value-based addressing

Where the initial RTI spec only allowed for type-based addressing, the RITN and JPSD projects [Calvin95, Powell96] both chose a value-based addressing approach. Data was addressed dependent on what value it currently had, as well as what type. This approach allows a federate host from getting swamped with 'all tanks' data, as only 'tanks within 10km of current_position' are received by that federate's host.

The only difficulty with value-based addressing is that it is heavily dependent on what the data means to the federation as a whole, and to each federate in particular. For example, what fields of bits represents the 'position' concept, and what does 'near my_current_position' mean? How fast do platforms move? Does that affect how often the addresses change? If they change too quickly, will the RTI have trouble adapting the multicast groupings? More analysis and implementation decisions must occur for a value-based addressing scheme that for a class-based scheme.

CLASSES OF ROUTING

Many types of routing exist overall -- we restrict our analysis to the use of multicast in a distributed simulation, where multicast is defined to be a single transmission of a packet for reception by multiple hosts. Note that two implementations of multicast exist: the IP protocol (where hosts join a multicast group for both transmission and reception), and ATM point to multipoint (where a source sends to an address consisting of multiple destinations). Each of the routing techniques outlined below may be implemented via either multicasting mechanism, however the implementation cost and resulting performance will vary. A detailed analysis of routing techniques may be found in [Powell96B].

As an object comes down from the application to the RTI, the RTI must have an algorithm to decide how to put that data onto the appropriate multicast group (i.e. route it). The RTI has *only three* basic pieces of information it can base the multicast group selection upon:

- The *Source* of the Data (which application or entity or unit this object comes from).

- The *Content* of the Data (the values of the attributes in the object, or information, like filter specs, derived from the values of the attributes).
- The *Destination* of the Data (the list of all those applications or hosts that have subscribed to this piece of data and for which it is destined).

Thus any multicast scheme can be categorized as one of:

- Source-based multicast.
- Data-based multicast.
- Destination-based multicast.
- Some combination of the above three.

As described in [Powell96], a number of differing approaches may be taken to utilizing network multicasting mechanisms within the filtering problem. The three basic approaches are summarized below. Note that the use of multicast groups (i.e. routing) is primarily independent of the mechanism used to determine the destination of data (i.e. addressing). Various addressing mechanisms may be used to make the routing decision for all classes of multicast.

Source-Based Multicast

Multicast addresses are assigned based on the source of a data item, i.e. every Data Source transmits its data on its own unique multicast group. Possible interpretations of "Source" are:

- Object ID (each object gets its own MC Group)
- Unit ID (a unit is a group of objects or entities, such as a platoon or company, each unit gets its own MC Group)
- Host ID (each computer gets its own MC Group)

Subscribers join the appropriate group to receive data.

Primary advantage: low numbers of multicast groups are required.

Primary disadvantage: efficient mapping of source to multicast group.

Data-Based Multicast

Multicast addresses are assigned based on the values of the full set of data items. The first idea (and most popular)used to date is *geographic* grids, where each object is transmitted on the MC group corresponding to the geographic region in which the object (entity) exists. Other options include the use of PDU (object) type, entity type, altitude, marking, etc.

Subscribers join the appropriate group to receive data.

Primary advantage: ease of implementation, no background traffic.

Primary disadvantage: large numbers of multicast groups are required for reasonable data segmentation.

Destination-Based Multicast

Multicast addresses are assigned based on the consumers of a given data item. Example:

Mcast Group 1: host_A, host_B, host_C

Mcast Group 2: host_A, host_B

Mcast Group 3: host_B, host_C

Mcast Group 4: host_A, host_C

As each data item is updated, a Destination List is produced (by any addressing scheme). MC groups are used to route data.

Primary advantage: filtering may occur at the source of data -- minimal loading on network, optimal addressing results in minimal disturbance of hosts.

Disadvantage: unacceptable number of MC groups required: 2^N -1, with *N* federates, to set up a fully-connected mesh.[iii]

CLASSES OF FILTERING

Filtering can be achieved at three places in a distributed system: data source, data destination, and the network (or in general, the transport mechanism).

· *Source Filtering*: helps eliminate unwanted traffic from getting onto the network and arriving at destinations unwanted. Potential big payoff.

· *Destination Filtering*: provides the least advantage to the overall system, since all three elements (source, network, and destination) are stressed to some extent, but is the most accurate mechanism.

· *Network Filtering*: In a distributed system, the primary mechanism for Network Filtering is the use of multicast: i.e. routing data only to consuming hosts.

All filtering techniques are used to produce a destination list -- i.e. what hosts require any given update_attribute, and what address will get it to them. Many different multicast schemes may be used to route the data, once the destination list is known. Four major approaches have been proposed to allow the inclusion of current-value information into the RTI routing decision[iv]. These are summarized below.

Categories

This scheme requires the federation to define a set of filters that sub-divide the potential values of attributes to be filtered on. Each filter defines a *category* that an attribute's current value falls within. Default categories are assigned by the RTI, based on

---

[iii] Unless address caching schemes or hierarchical architecture techniques are employed. Natural hierarchy point for multicasting: LAN/WAN boundary.

[iv] An interesting approach used in the NSS design was discussed, but was not formally proposed.

the already known class/attribute type information from the FOM.

The standard model for RTI interaction still holds: the RTI is required to deliver only those update_attributes that meet a federate's current subscription list. If the federation defines its own set of categories, individual federates subscribe to categories of data, not to class/attribute sets. The RTI interaction model does not change -- the RTI now simply delivers update_attributes based on the user-supplied categories.

To expand on our example federation, tank objects now consist of two attributes: position and status. Position remains an (x,y) coordinate, and STATUS is a boolean variable: LIVE/DEAD.

Examples of categories for this federation depend on what is deemed important. If there are few tanks planned in the federation execution, then the default assignment of categories by the RTI is sufficient. If a federate is not interested in DEAD TANKS, then the federation-defined categories would consist of:

category 1: type == TANK, status == LIVE
category 2: type == TANK, status == DEAD
category 3: type == TANK

Any given federate could then subscribe to one of the three categories of data, and be assured that objects which match the other two categories will not be delivered to its host. Thus in the example 5,000 tank scenario, if 2,000 of the tanks are dead, and federate_A subscribed to category_1 data, 'only' 3,000 update_attributes are received.

To further improve performance, the federation could extend the categories to:

c 1: type == TANK, status == L, pos.x <= 50
c 2: type == TANK, status == L, pos.x > 50
c 3: type == TANK, status == D
c 4: type == TANK

In this example, we assume a (x,y) battlespace ranging from (0..100, 0..100). Via the above categories, the federate has broken the battlespace into two playboxes:

(0..50, 0..100) and (51..100, 0..100)

If federate_A currently represented an entity at position (10, 0) (i.e. the left playbox), it would then subscribe to category_1 data. Thus in the example 5,000 tank scenario, if 2,000 of the tanks are dead, and 1/2 of the remaining LIVE tanks are in the other playbox, only 1,500 update_attributes are received by federate_A.

The above example may be continued by further sub-dividing the battlespace into smaller playboxes (or sectors), or using further attributes of the objects. The intent is to allow the federation to use federation-specific knowledge of the data to break the set of attribute values down into a finite set of categories. Each federation may specify the amount of categorization it feels is necessary to achieve its performance goals: from simply using the default class/attribute type categories created by the RTI, to a full geographic sectorization scheme and discrete value changes of key attributes.

Note the very specific knowledge used by the federation's categorization scheme. The federation has used the fact that the status attribute has only two possible values, and that some federates never require data about DEAD tanks. The federation has also used the knowledge that the first attribute of the tank object is used to denote position, and that federates require different data to be delivered, based on the current value of all object's position attribute.

Data Publication: When a federate changes a value of an attribute, it then evaluates that attribute against the set of categories. The federate then passes the new value to RTI with the category number which currently describes the attribute.

Data Subscription: Federates examine the list of available categories, and subscribe to the categories which describe data that they require. Federates may change what categories they are subscribed to at any time.

Multicast Optimization: multicast groups are mapped one per category. This use of multicast is considered *data-based* multicast, where the value of transmitted data is used to determine what multicast groups are transmitted to and received from.

Source-based Filtering

Source-based filtering describes a class of addressing used by the JPSD system, as well as a number of distributed database systems . In this approach, the filters created by each federate are placed at all potential sources of data in the distributed system. As data is changed, it is run through the filters on a simple pass/fail basis. Each filter also contains the network address of the federate who created the filter. Data is thus only routed to exactly the set of federates who require it.

While on the surface a very appealing approach, a number of implementation issues arise to do with efficient distribution of filters and subsequent changes to filters. While not inconsiderable, efficient solutions to these problems are relatively known, and entirely the responsibility of the federate, not the RTI. The reader is referred to [Powell96], where an 8,000 to 10,000 DIS-entity exercise is detailed. Further work on the scalability of this approach may be found in the STOW runtime architecture implementation notes (www.stow.com). Also note the applicability of hierarchical improvements to the basic source-based filtering approach. Other improvements, such as unification of filters, ordered comparisons, and limited distributed of filters are also outlined in the STOW architecture documents.

Data Publication: When a federate changes a value of an attribute, it then evaluates that attribute against the local set of filters (created by remote federates). If the new attribute value passes a filter, the federate then transmits the new value to the list of federates attached to that filter.

Data Subscription: Federates create a set of filters which describe the types and values of attributes they are interested in. These filters are sent to any federate capable of producing such types of data.

Multicast Optimization: Each multicast group consists of a list of federates to whom data of any type or value is to be sent. For example, multicast group 1 would consist of fed_A, fed_B, fed_C,

multicast group 2 would be fed_A, fed_B, group 3 == fed_A, fed_C, group 4 == fed_B, fed_C. This use of multicast is considered destination-based multicast, where the destination of the data is used to determine what multicast groups are transmitted to and received from.

Filter Space

Filter space is a novel approach to obtaining sufficient information from the current value of data items to make effective routing decisions without specific knowledge of the data's type, value, or meaning to the federation. This enables a filtering mechanism to be built into the RTI that is capable of supporting many different federations without minimal coding required by the federation.

Filter space requires the federation to create a N-dimensional space, each dimension corresponding to an attribute to be used in the filtering decision. For example, a filter spec might be created with five dimensions: lat, long, height above ground, RCS, and velocity. Each dimension is normalized (or set with ranges now?).

Data Publication: When a federate changes a value of an attribute associated with a filter spec, it then evaluates where in filter space the five-attribute data set (lat,long,HAG,RCS,vel) exists[v]. The federate then publishes the new value via the RTI, with the abstracted point in filter space that is associated with the data set.

Data Subscription: Similar to the publisher, potential subscribers create a N-dimensional filter spec, and subscribe to an abstract region within that filter space. When the RTI detects a data set being published that falls within the declared region, it brings the matching data to the subscribing federate.

Multicast Optimization: multicast groups are mapped to 'cells' within filter space. All data sets whose abstracted point in filter space falls within cell 1 are transmitted by the RTI on multicast group 1. RTI Ambassadors whose local federate has a region of declared interest within cell 1 will then join multicast group 1 and receive relevant data for the federate. This use of multicast is considered *data-based* multicast, where the value of transmitted data is used to determine what multicast groups are transmitted to and received from. While the Filter Space API does not exactly *preclude* a source- or destination-based scheme underneath, it is weighted heavily towards a data-based approach.

Synopsis

The synopsis approach requires each federate to maintain a very approximate representation of all attribute values it controls. This *synopsis* is then distributed to all other federates in the system. Each federate then examines it local copy of the global synopsis to determine which attributes might be relevant. For any attributes that might be relevant, the federate then requests that the RTI begin delivering the exact values of that specified set of attributes.

For example, the example federation above may be expanded to include a low-resolution position attribute. Thus the TANK object contains: position, status, low_res_position. As in the category example, LOW_RES_POSITION may simply be left-playbox or right-playbox. An example federate would have subscribed to the low_res_position of all TANKS. Based on the (locally known) fact that local tanks are in the left-playbox, the federate would then subscribe to the high-resolution position of all tanks whose low_res_position is equal to left-playbox.

Data Publication: Federates each maintain a 'synopsis' of the current set of attributes they produce. The synopsis consists of low-resolution views of the attributes. Synopsis updates are occasionally updated to all other federates. When a federate changes a value of an attribute, it sends an update of that attribute to all federates who have subscribed to it (see below).

Data Subscription: Federates regularly examine the low-resolution synopsis of available data. For attributes that may be of interest, the federate explicitly subscribes to the high-resolution version of that attribute.

Multicast Optimization: either source or destination based multicast may be used. An additional side feature exists: Wide-Area Viewers can make use of the low fidelity information with no further network loading.

IMPLEMENTATION CONSIDERATIONS

Mapping Alternate Filtering Schemes to the Filter Space API

It has been proposed within the AMG that the filter space API is flexible enough to support alternate filtering mechanisms without significant loss of performance. The suggested approach is based on exploiting knowledge of the RTI's internal implementation of filter space to obtain direct control over multicast allocation. The approach is summarized below.

Assume a federation defines a one-dimensional filter space, and assigns a range of 1..10 to that dimension. Using the *grid* feature of the API, the federation may break the filter space into 10 cells. Given that the current implementation of the RTI assigns filter space cells on a one to one basis to multicast groups, the federation may now use addressing mechanisms (i.e. filters) other that filter space and be able to route the data onto multicast groups via the RTI (see also: *Implementation of Routing Options*, below).

The above approach would seem to provide a prima facie solution to supporting other filtering techniques via the filter space API. However, two significant drawbacks exist.

The first drawback is that the solution is entirely dependent on the RTI's internal implementation of

[v] An optimization may be done, where the federate used *thresholds* to determine if the attribute's location in filter space need be checked.

filter space. If the RTI is to be considered a 'black-box' component which provides a given set of functions via any number of internal implementation options, this approach is considered inadequate. If the federation has access to, and control over, the RTI's internal implementation, then this approach, while semantically poor, may be deemed reliable[vi]. Note that the proposed 'commercialization' of the RTI (with multiple RTI implementations and vendors), renders the assumption of internal implementation knowledge potentially unreliable, and may reduce the ability of federations with specialized filters to use multiple RTI implementations.

The second drawback concerns the loss of semantic content via using the API in a completely different manner than it was designed to support. In particular, the functionality performed by the RTI (routing) does not match the functionality defined by the API (filter space). While this mis-match is not a 'show-stopper', such distortions of an API are a cause for concern. To use an extreme example, one could argue that the basic object-based subscription provided by the API is sufficient for federation-tailored routing. A federation could simply define a set of classes which conceptually describe multicast groups, thus controlling routing by subscribing and unsubscribing to various objects. Few would argue that such a scheme would be a poor use of the object subscription mechanism, yet it would function.

Also of potential concern is the ability of the federation to implement hierarchies of filters within the network structure. As outlined earlier, there may be substantial improvements possible from implementing filters (possibly of different types) at various points in the network hierarchy, most notably at the LAN/WAN boundaries. While such filters may be created internal to the RTI, a federation will have little to no control over the types and locations of filters without the ability to modify the RTI.

Implementation of Routing Options

Internal to the RTI: While filter space most naturally maps to a data-base use of multicast, it is certainly possible to implement any of the basic multicast routing algorithms within the RTI. It is likely that the optimal routing technique will vary across federations, with considerable overlap between federations which exchange data in similar patterns.

Via the Filter Space API: If the one-dimensional filter space approach outlined above is used, then all multicast routing approaches may be built above the RTI, if the RTI implementation is a 'glass-box' -- i.e. the user is aware of how the component is built, but may not modify it.

ANALYSIS CONCLUSIONS

While the majority of the RTI's functional specifications are sufficiently defined and tested to be included in the DoD-wide HLA standard, some risk remains with the Data Distribution Management (DDM). The proposed scalability approach to DDM, filter space, appears sufficient to support data-based filtering schemes across a range of federations. Further, the use of other filtering schemes layered on top of the filter space API is possible. However, significant loss of semantic clarity and potential performance losses exist if such layering occurs. Federations must also have implementation-level knowledge of the RTI's internal routing algorithms. Given that little experimental data exists to support one filtering scheme over another, a more generic interface to DDM than filter space would be preferable from an architectural design perspective. However, from a use perspective, there is considerable value to having a common approach to filtering across federations. From that perspective, and that the only standardization approach available at the moment is the HLA, filter space may be suitable for the initial RTI specification.

DDM risk is considerably lessened if two assumptions apply:

- Federations have the ability to 'tailor' the internal implementation of the RTI to better meet their purpose (i.e. the RTI is a component under the federation's control, and not a black-box of functionality). In particular, can the RTI's internal implementation of routing be modified.[vii]
- The RTI API is flexible -- i.e. a more generic interface may be added at a later date if experiments with other filtering schemes show promise.

ACKNOWLEDGMENTS

The author would like to thank Ed Powell, Darrin West, Jesse Aronson, Mike Mazurek, Glenn Tarbox, Jim Watson, and Tim Eller for many profitable discussions about the role of multicast and filtering in distributed simulation. Many of the ideas discussed in this paper were the result of collaboration between the author and these individuals and their contributions are gratefully acknowledged.

The definition of filtering techniques and participation in the AMG filter working group was primarily accomplished under the STOW SEID program.

Further thanks are due to Jim Cantor for keeping the author on travel long enough to write this paper,

[vi] Also note that any internal optimizations done dynamically by the RTI to improve multicast performance are unlikely to increase performance, as their optimization heuristics must, by definition, be based on the characteristics of filter space. Given that alternate addressing schemes and/or routing approaches are mapped to the filter space API, the characteristics are likely to be substantially different -- in this case, internal RTI attempts to dynamically improve multicast performance may well slow down the system.

[vii] As stated earlier, allowing tailoring may introduce standardization and compliance risks.

and to USAir, whose flights average one battery charge.

REFERENCES

Calvin, J., J. Seeger, G. Troxel, and D. Van Hook. 1995. STOW Realtime Information Transfer and Networking System Architecture. In *Proceedings of the 12th DIS Workshop*.

Chaiken, D., J. Kubiatowicz, and A. Agarwal. 1991. LimitLESS Directories: A Scalable Cache Coherence Scheme. *Communications of the ACM*, Reference # 0-89791-380-9/91/0003-0224.

Powell, E. "The Use Of Multicast and Interest Management in DIS and HLA Applications," Proceedings of the 15th DIS Workshop.

Powell, E. *et al.*, "Joint Precision Strike Demonstration (JPSD) Simulation Architecture," Proceedings of the 14th DIS Workshop, IST-CR-96-02, 3/96.

Mellon, L., "Hierarchical Filtering in the STOW System," Proceedings of the 14th DIS Workshop, IST-CR-96-02, 3/96.

Steinman, J., "Declaration Management in HLA". *Draft* Version 0.1.

AUTHOR BIOGRAPHY

LARRY MELLON is a senior computer scientist and branch manager with Science Applications International Corporation (SAIC). He received his B.Sc. degree from the University of Calgary. His research interests include parallel simulation and distributed systems. He is a lead architect for the ARPA-funded Synthetic Theater of War (STOW SEID) Program.